

MANAGING SYSTEM RESOURCES

After reading this chapter and completing the exercises, you will be able to:

- ♦ Access the `/proc` file system to learn about system status
- ♦ Use the `ps` and related commands to control how processes use system resources
- ♦ Track physical and virtual memory usage
- ♦ Locate and relieve system bottlenecks

In the previous chapter you learned about safeguarding information on your Linux server by creating a disaster plan for your information systems. You learned about using an uninterruptible power supply and checking the integrity of your file systems using the `fsck` program. You also learned about several fault tolerance capabilities that you can add to Linux, such as RAID disk drives.

In this chapter you learn how to manage system resources so that users on your Linux system can complete their work effectively. Among other resources, you will learn about CPU time (processing capacity) and system memory. Once you are familiar with the commands that manage these resources, you will be able to locate and alleviate bottlenecks that restrict system performance.

ACCESSING THE `/proc` FILE SYSTEM

In any operating system, many activities are taking place that are not visible on your computer screen. This is especially true of a Linux or UNIX system, in which multiple background processes (daemons) continue working even when users do not appear to be running applications. Background activities include managing network traffic, sending files to printers, logging Linux kernel activity, and managing system memory utilization (such as moving information to and from the swap partition).

When necessary, you can use the **`/proc` file system** to see what the operating system kernel is doing at any given time. This feature is called a file system because you access information in `/proc` the same way you do in any file system—by reading a file using a command such as `cat`. But the information in the `/proc` file system is not stored on a hard

disk. When you view the contents of a filename in `/proc`, the Linux kernel responds with live information about the status of a process, memory, or other items related to the kernel or other system resources. This information changes from moment to moment inside the Linux kernel.

For example, `/proc` can provide a list of the amount of memory on your system and how it is being used. Execute the command `cat /proc/meminfo` to show memory information. Sample output for this command is shown here:

```

            total:      used:      free:      shared:  buffers:  cached:
Mem:        31559680   30765056  794624    45461504  782336    12251136
Swap:       133885952  1318912   132567040
MemTotal:   30820 kB
MemFree:    776 kB
MemShared:  44396 kB
Buffers:    764 kB
Cached:     11964 kB
SwapTotal:  130748 kB
SwapFree:   129460 kB
```

You can also write information to some filenames in `/proc`. For example, you can write a value to `/proc/sys/fs/file-max` to change the number of file handles that can be used at one time in Linux. (A **file handle** is an internal storage mechanism that allows a single file to be opened and used in Linux.) This command displays the number of file handles currently configured in the Linux kernel:

```
cat /proc/sys/fs/file-max
```

This command changes the number of file handles available in the Linux kernel to 2048:

```
echo 2048 > /proc/sys/fs/file-max
```

Most files in `/proc` are only used to display information about the kernel. The examples shown so far illustrate the types of information you can see by viewing files in the `/proc` file system. The next two sections explain how to use `/proc` to view a great deal of information about your system hardware and the processes running on Linux.

Viewing Device Information

The `/proc` file system provides information about many parts of your system hardware. This information can be useful as you configure devices or software services. Table 10-1 lists the paths in your Linux directory structure where you can access various hardware information.

Table 10-1 Hardware Information Accessible Through /proc

Hardware information	Path
Battery information for systems using advanced power management (APM) software	/proc/apm
CPU information	/proc/cpuinfo
Direct memory access (DMA) channels used by system devices	/proc/dma
Interrupts configured for system devices	/proc/interrupts
Ports (memory addresses) used to communicate with system devices	/proc/ioports
File systems currently available to the Linux kernel	/proc/mounts
Disk partitions known to the Linux kernel	/proc/partitions
Information on all PCI devices in your system, such as video cards and hard disk controllers	/proc/pci
Information on all SCSI devices in your system	/proc/scsi and its subdirectories
Information on the real-time clock in your system	/proc/rtc
Swap device information	/proc/swaps

To see the hardware interrupts used by your computer, use this command: `cat/proc/interrupts`. On a typical Linux system, this command would produce the following output:

```

          CPU0
0:         149607      XT-PIC  timer
1:          2589      XT-PIC  keyboard
2:              0      XT-PIC  cascade
4:         1431      XT-PIC  serial
5:              1      XT-PIC  pcnet_cs
8:              2      XT-PIC  rtc
12:           2      XT-PIC  PS/2 Mouse
13:           1      XT-PIC  fpu
14:        78763      XT-PIC  ide0
15:           4      XT-PIC  ide1
NMI:           0
```

When you add devices to your system or configure existing hardware, you can use the output from the preceding command to verify that two devices are not configured to use the same interrupt number, or that a device such as a sound card is not configured to use an interrupt that is actually assigned to another device.

Keep in mind that you won't find all of the hardware information required for configuring Linux systems in /proc. For example, if your system includes a PCI video card, /proc/pci will not provide sufficient information to allow you to configure the X Window System and a graphical desktop. Video card details, such as the video chipset, are not used by the kernel, so they are not maintained in the /proc file system.

Many of the Linux system administration utilities rely on information from `/proc`. One example is the set of system information provided by the Control Center in the KDE Desktop, which is shown in Figure 10-1. Notice that the list of items under Information corresponds quite closely to the list of hardware information described in Table 10-1.



Figure 10-1 System information in the KDE Control Center

Other utilities such as `free` and `ps`, described later in this chapter, also obtain information from `/proc` and present it in a format that is easier to read than are the `/proc` files themselves.

Viewing Process Information

The `/proc` file system contains information for each process. This information is updated moment to moment, as the status of a process changes. Before you can access information in `/proc` regarding a specific process, you need to find the process's PID number. (As you learned in Chapter 9, every process running on Linux is assigned a process ID, or PID, number.) To find the PID for a running process (such as a program that you have started), you need to use the `ps` command.

Suppose that, using the `ps` command, you discover the PID for a particular process is 1066. You can then look for information about the process in a directory named `/proc/1066`. Much of this information is difficult to use directly because it consists of many numbers without explanations—hence the need for administrative utilities that can display process

data in a meaningful way. One part of the process data that is readable, however, is the `cmdline` file, which tells you the command used to start the process. The following will tell you the command used to start the process with a PID of 1066: `cat /proc/1066/cmdline`. This produces the following output, which indicates that the `enlightenment` command with two parameters (`-clientId` and `default2`) was used to start the process.

```
enlightenment -clientId default2
```

You can explore other subdirectories of a process information directory like `/proc/1066` on your own Linux system to determine what types of information about a process are available to you and to Linux utilities.

MANAGING PROCESSES

In Chapter 9 you learned about using the `ps` command to send signals to processes running on Linux. In this section you will learn more about the output of the `ps` command. You will also learn how you can control the way processes use system resources on Linux.

To effectively manage your system, you need to select the processes you're interested in and then display various pieces of information about those processes. You can use several options in the `ps` command to select which processes are included in the command output. Table 10-2 summarizes these selection options.

10

Table 10-2 `ps` Options Used to Select Processes Included in Command Output

Command-line option	Description
<code>-A</code>	Selects all processes on the system.
<code>T</code>	Selects all processes running in the current terminal.
<code>x</code>	Selects all processes that were not started normally from a terminal (this list includes system initialization scripts and network services).
<code>r</code>	Restricts output to running processes (those that are not sleeping). This option is used in conjunction with another selection option.
<code>-C</code>	Selects processes by the command used to start the process. To use this option, you need to follow it with the name of the command.
<code>-p</code>	Selects processes by PID number. To view information on a single process, enter its PID number as a value after the option.
<code>--user</code>	Selects processes by username. To use this option, type the username after the option.
<code>--group</code>	Selects all processes belonging to users who are members of the group named after the option.

For example, `root` could employ the following command to list all processes owned by user `jsmith`:

```
ps --user jsmith
```

The `ps` command can provide many types of information about each process. You can select which pieces, or fields, of information `ps` includes in output by using command-line options. Table 10-3 shows the information available from the `ps` command for each process. Many of the terms used in the Description column (such as “nice level”) are discussed at length later in this chapter, but a few are used only for special troubleshooting tasks and are beyond the scope of this book.

Table 10-3 Process Information Fields Available from `ps`

Display code (column heading in <code>ps</code> output)	Description	Command-line option
PID	Process ID	<code>pid</code>
PPID	Process ID of the parent process	<code>ppid</code>
PGID	Process group ID	<code>pgid</code>
SID	Session ID	<code>sess</code>
TTY	Controlling terminal	<code>tty</code>
TPGID	Process group ID of the owner of the terminal running the process	<code>tpgid</code>
USER	Owner of the process	<code>user</code>
PRI	Time left of a possible timeslice allocated to the process	<code>pri</code>
NICE	Nice level	<code>nice</code>
PLCY	Scheduling policy	<code>plcy</code>
RPRI	Real-time priority	<code>rpri</code>
MAJFLT	Number of major faults loading information from a file system	<code>majflt</code>
MINFLT	Number of minor faults (with no disk access involved)	<code>minflt</code>
TRS	Size of the text used by the program (in KB)	<code>trs</code>
DRS	Size of the data used by the program (in KB)	<code>drs</code>
SIZE	Virtual image size of the process (in KB)	<code>size</code>
SWAP	Space used on swap device by this process (in KB)	<code>swap</code>
RSS	Kilobytes of the program resident in memory	<code>rss</code>
SHARE	Shared memory size in KB	<code>share</code>
DT	Number of pages of information that are dirty (not yet updated to hard disk)	<code>dt</code>
STAT	State of the process	<code>stat</code>
FLAGS	Process flags	<code>f</code>
WCHAN	Kernel function at the point where the process is sleeping	<code>wchan</code>
UID	User ID of the owner of this process	<code>uid</code>

Table 10-3 Process Information Fields Available from `ps` (continued)

Display code (column heading in <code>ps</code> output)	Description	Command-line option
<code>%WCPU</code>	Weighted percentage of CPU time consumed	<code>wpcpu</code>
<code>%CPU</code>	Percentage of CPU used since last update	<code>pcpu</code>
<code>%MEM</code>	Percentage of memory used	<code>pmem</code>
<code>START</code>	Time that the process was started	<code>start</code>
<code>TIME</code>	Total amount of CPU time (cumulative) that the process has used since it was started	<code>time</code>
<code>COMM</code>	Command line that started the process (abbreviated)	<code>comm</code>
<code>CMDLINE</code>	Command line that started the process (complete)	<code>cmd</code>

The options from Table 10-2 are generally combined with output selection options from Table 10-3. To use a command-line option from Table 10-3 to view selected process information, use the arguments given in the third column of the table after the `o` option. For example, to display all processes (option `-A`), including processes without controlling terminals (option `x`), and display only the command line for each of those processes (option `comm`), use this command:

```
ps -A xo comm
```

You can use additional `ps` options to display a predefined set of process information. Table 10-4 describes these options.

Table 10-4 Predefined Sets of Process Information Displayed by the `ps` Command.

Command-line option	Description
<code>j</code>	Show fields related to controlling jobs in a shell
<code>s</code>	Show fields related to signals that each process handles
<code>u</code>	Show fields that define how the owner of each process is using system resources
<code>v</code>	Show fields detailing how each process is using virtual memory
<code>l</code>	Show numerous fields considered by system administrators to be of interest in tracking processes but not otherwise related (as the above groupings for job control, virtual memory, and so forth, are groups of related fields)
<code>o</code> or <code>--format</code>	User-defined format (display all of the fields listed after the option; each field to be included in the output is defined by using a code from the right column of Table 10-3)



The man page for the `ps` command contains additional details about how to use `ps` command options.

As a system administrator, you have the task of managing numerous processes started by numerous users. You need to track how these processes consume system resources—particularly CPU time. The `%CPU` and `TIME` fields of the `ps` command output are especially useful in tracking how processes are using CPU time.

The `%CPU` field compares the amount of CPU time used by a process with the total time elapsed since the previous computation of the `%CPU` field in the form of a percentage. The Linux kernel tracks a small slice of time (such as one second) and then determines for how much of that one second a process was using the CPU. That computation creates a percentage used for the `%CPU` field of the `ps` command output. The `%CPU` field does not show the average amount of CPU time used by the process since it was started. The `%wCPU` field is weighted to show a 30-second average of the percentage of CPU time used by a process. This field is more helpful for showing the overall usage pattern for a process.

The `TIME` field provides a cumulative measure of the amount of CPU time consumed by a process. Processes that have been running since the system was booted may still have a `TIME` value of `0:00` because they are background processes with very little activity to monitor. But some processes that are only recently started may show a large `TIME` value (for example, `5:30` to indicate five and a half minutes), indicating that they are using a lot of the CPU's time. The `START` field tells you when a process began running. When a process has a large `TIME` value after running for only a short time, you may need to change the priority of that process to prevent it from slowing down other processes. The next section describes how to do this.

Process Priorities

Each process is assigned a **priority** when it is started. This priority determines how much CPU time is granted to the process to complete its tasks. Normally, all processes have the same priority—that is, all processes are assigned an equal portion of CPU time for processing. Another name for the priority of a process is **nice level**.



`PRI` in the `ps` command output is short for *Priority*, but this field actually indicates how a process is using the CPU time allocated to it. You will notice that this field changes regularly for a given process as the process works, using CPU time at different moments.

The `NICE` or `NI` field in the output of the `ps` command indicates the nice level, or priority level, assigned to a process. The nice level is a fixed value for a process. This value determines whether a process receives extra CPU processing time or less CPU processing time (compared to other processes running on the system). Thus the nice level can be set to change the relative priority of a process running on Linux. The idea behind the name “nice” is that if a user on the system decides a certain program is not time sensitive, the user can make the program “nicer” to other users’ programs by giving up some of its CPU time.

Changing Priorities with **nice** and **renice**

You can alter the priority of a process by using the **nice** command or the **renice** command. The standard nice level is 0, meaning that all processes start with equal priority. Any user can raise the nice level of a process that he or she has started (and thus owns), making it nicer to other programs. The highest nice level, which makes a program run the slowest, is 20. The **root** user can make any process nicer, but **root** can also make programs less nice by lowering the nice level. The **root** user can lower the nice level of a process to -20, which gives that process a lot of extra CPU time. The current nice level of a process is shown as the **NI** or **NICE** field in the output of the **ps** command.

The **nice** command is used when launching a process to assign a nonstandard priority to that process—to make it nicer (or less nice if you are **root**). The parameters used with the **nice** command include a nice level and the name of the program to launch. For example, to start a script named **analyze** with a nice level of 10, use this command:

```
nice -10 analyze
```

The **renice** command is used to change the nice level of a process that is already running. Regular users can change the nice level of processes that they own; **root** can change the nice level of any process. The **renice** command requires at least the PID of the process you want to affect. The **root** user can perform more complex tasks using **renice**. These tasks require additional information such as the user ID (UID) of the owner of a process. For example, suppose you start a complex script named **analyze**. After starting the script, you decide you can wait for the results of the script, allowing other programs to run more efficiently. You can change the nice level of the process by using the PID of the running script with the **renice** command. For example, if the PID of the **analyze** script is 1776, this **renice** command will change the script's priority.

```
renice +10 1776
```

Suppose that as the system administrator you discover that a certain user is running several computationally intensive programs that are slowing down system response for other users. After checking with this user, you learn that the programs are a valid use of the system resources, but they are not time critical. To make things run more smoothly for other users, you change the priority of all processes run by that user so they run more slowly. When the intensive programs are finished running, you can reset the user's priority level back to a default value (0). This command changes the priority of all processes owned by user **jsmith**:

```
renice +5 -u jsmith
```

Examples of possible uses for **renice** include:

- The system administrator requests that all noncritical processes be run at higher nice levels while certain intensive system administration tasks are being completed.
- A user has started several processes; some are time sensitive, some are not. By raising the nice level of the processes that are not time sensitive, the other processes should be completed sooner. (The status of processes owned by other users affects the results of this effort, however.)

Viewing Processor Usage with `top`

The `top` utility is a standard command-line utility included with all versions of Linux. You can use `top` to display a list of processes on your Linux system similar to the list shown by `ps` with the `aux` options. But the output of `top` is arranged by how much CPU time is being used by a process. The process that is consuming the greatest amount of CPU time is shown at the top of the list, and so forth, to the bottom of the list. The output of `top` is also updated regularly (every five seconds by default, although you can configure the update interval yourself).

Running `top` and then leaving it visible on your screen allows you to watch the activity of different processes to see which ones are using a lot of CPU time. If one process begins to take more than its fair share of CPU time (in your judgment as system administrator), you can immediately take corrective action by changing the nice level of that process within the `top` command, as explained later in this section.

`Top` is normally launched without any options, like this:

```
top
```

As `top` starts running, it takes over the text window that you are working in. You cannot run `top` in the background (using the symbol `&` after the command), because `top` sends its output immediately to the screen. You can, however, use redirection (such as the `>` operator) to send the output of `top` to a file. Figure 10-2 shows how the output from `top` appears in a terminal window.

```

4:53pm up 9:54, 4 users, load average: 0.54, 0.32, 0.21
87 processes: 85 sleeping, 2 running, 0 zombie, 0 stopped
CPU states: 3.5% user, 11.4% system, 0.7% nice, 84.2% idle
Mem: 30820K av, 29492K used, 1328K free, 19220K shrd, 656K buff
Swap: 130748K av, 20288K used, 110460K free, 10404K cached

  PID USER   PRI  NI  SIZE  RSS SHARE STAT   LIB %CPU %MEM    TIME COMMAND
 1382 root    14   0 1804 1508   948 S    0   7.6  4.8   1:37 gtop
   897 root    12  10 1892 1204   856 S N    0   1.7  3.9   9:31 kpm
 1367 nwell    3   0   240   220   172 S    0   1.7  0.7   0:22 vmstat
 1411 root     3   0   512   472   312 S    0   1.5  1.5   0:15 top
 1427 root     4   0 1024 1024   824 R    0   1.3  3.3   0:00 top
   703 root     5   0 10820 8348  1416 S    0   0.7 27.0 23:46 X
   721 root     1   0 1264 1024   608 S    0   0.3  3.3   0:11 enlightenment
 1320 root     1   0   468   304   208 S    0   0.1  0.9   0:00 xscreensaver
 1423 root     0   0 3016 3016  2448 R    0   0.1  9.7   0:00 gnome-terminal
     1 root     0   0   100    52    36 S    0   0.0  0.1   0:04 init
     2 root     0   0     0     0     0 SW    0   0.0  0.0   0:00 kflushd
     3 root     0   0     0     0     0 SW    0   0.0  0.0   0:00 kpiod
     4 root     0   0     0     0     0 SW    0   0.0  0.0   0:01 kswapd
     5 root    -20 -20     0     0     0 SW<   0   0.0  0.0   0:00 mdrecoveryd
   102 root     0   0   188   160   128 S    0   0.0  0.5   0:00 apmd
   216 bin     0   0     68     0     0 SW    0   0.0  0.0   0:00 portmap
   263 root     0   0   216   152   124 S    0   0.0  0.4   0:00 syslogd
   274 root     0   0   364     0     0 SW    0   0.0  0.0   0:00 klogd
   288 daemon    0   0     72     0     0 SW    0   0.0  0.0   0:00 std
   302 root     0   0   164   112    84 S    0   0.0  0.3   0:00 crond
   315 root     0   0   120     0     0 SW    0   0.0  0.0   0:00 cardmgr
   403 root     0   0     84     0     0 SW    0   0.0  0.0   0:00 inetd
   421 root     0   0   928   456   368 S    0   0.0  1.4   0:00 named
   435 root     0   0     68     0     0 SW    0   0.0  0.0   0:00 lpd
   440 root     0   0   100     0     0 SW    0   0.0  0.0   0:00 lpd

```

Figure 10-2 Output from the `top` command

You can access many options from the keyboard as you view the output of `top`. One of the most useful is the ability to renice a process. To renice a process in `top`, press the `r` key and enter the PID of the process. For example, suppose the top few lines of the process list in `top` look like this:

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	LIB	%CPU	%MEM	TIME	COMMAND
1066	jsmith	17	0	1012	1012	820	R	0	4.7	3.2	0:00	analyze
1	root	0	0	100	52	36	S	0	0.0	0.1	0:04	init
2	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kflushd
3	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kpiod
4	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kswapd

To change the nice level of process 1066 (the `analyze` command, as indicated by the far right column), you would follow these steps:

1. Press the `r` key. A message appears above the process list asking you to enter the PID of the process to be reniced.
2. Enter the PID of the process (1066 in this example).
3. A message appears asking you for the new value to assign to this process. For this example, the nice level is being raised to 10; it could also be lowered from 0 to a negative number if you are running as root. Enter 10.
4. Watch the `NI` column of the process listing to see the nice level value change. Because of the higher nice level, the process moves down in the process list after a moment as its CPU usage decreases.



You will often see the `top` command itself in the output of `top`. When the `top` program is listed near the top of the output, you can be sure that the system is not under a heavy load.

As you are viewing the output of the `top` command, you can use the keys listed in Table 10-5 to control `top`. Other command options can also be specified on the command line when you first launch `top`.

Table 10-5 Interactive Commands in `top`

Description	Press this key	Notes
Update the process list display immediately.	Spacebar	
Show a help screen with a command listing.	h or ?	
Kill a process.	k	You will be prompted for the PID.
Change the number of processes included in the display.	n or #	You will be prompted for the number of processes to include.
Quit the <code>top</code> program.	q	
Renice a process.	r	You will be prompted for the PID and new nice level.
Change the automatic update interval.	s	You will be prompted for a value (in seconds) for the update interval.

Additional options are available for sorting information in `top`, displaying or hiding certain information fields, and changing how some fields (such as %CPU) are calculated. See the man page for the `top` command for further details.

Using Graphical Process Management Tools

Several graphical process management tools are available for Linux. This section briefly describes where to find some of those tools and how to use them.

The `kpm` utility (for KDE Process Manager) is included with the KDE Desktop. It is a fairly new utility, so it may not be included on your Linux system if you are using an older version of KDE. In this case, you can download `kpm` from the KDE Web site at <ftp://ftp.kde.org>. On most KDE installations you can start the `kpm` utility by selecting **Process Management** from the **Utilities** or **System** menu. The main window of `kpm` is shown in Figure 10-3.

The process list in `kpm` is similar to the output of the `top` command. Fields of information are shown for each process. The list is updated every few seconds. Above the list of processes in `kpm` is a set of graphs that show the system's CPU load and memory usage. (Other CPU load utilities are described in the next section.)

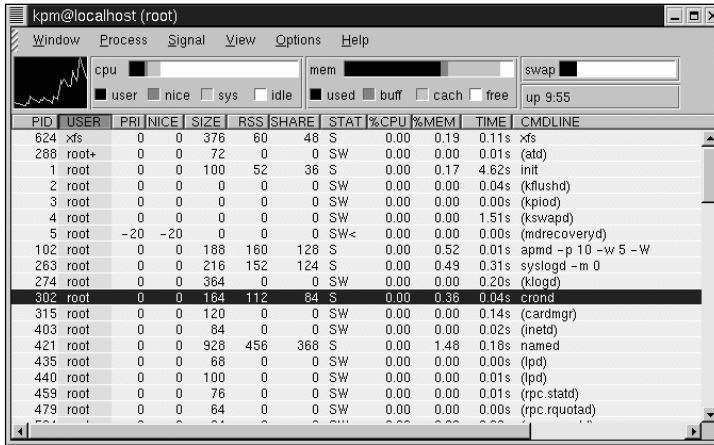


Figure 10-3 Main window of the `kpm` process management utility

Among other things, `kpm` allows you to:

- Click on system information graphs to change them to numeric data, and vice versa. Graphs are easier to read quickly, but numbers are more precise.
- Click on a process to select it, and then choose `Renice` from the `Process` menu to change the nice level of the selected process.
- Click on a process to select it, and then use the `Signal` menu to terminate, kill, or restart (that is, hangup) the selected process.
- Right-click on any process to open a menu of the most commonly used signals (such as `Terminate` and `Kill`). This menu also gives you access to the `Renice` option, which opens a dialog box in which you can change the nice level of the selected process.
- Use the `view` menu to select which fields of information to include in the list of processes. The `Select Fields` item on the `view` menu lets you choose exactly which information to include for each process.
- Click on a column heading (such as `PID` or `TIME`) to sort the list of processes by that field. You can click a second time on the same field to reverse the sorting order.



A program called **ktop** is included with some Linux systems. It runs on KDE and is similar in many ways to `kpm`.

The `kpm` utility is designed to run on Linux systems that employ the KDE Desktop. On systems (such as Red Hat Linux) that use the Gnome Desktop, you can use the **Gnome System Monitor utility**. The Gnome System Monitor, which is available on the Utilities menu, provides functionality very similar to the `kpm` program. Figure 10-4 shows its main window. To start the Gnome System Monitor from a command line, use the command `gtop`.

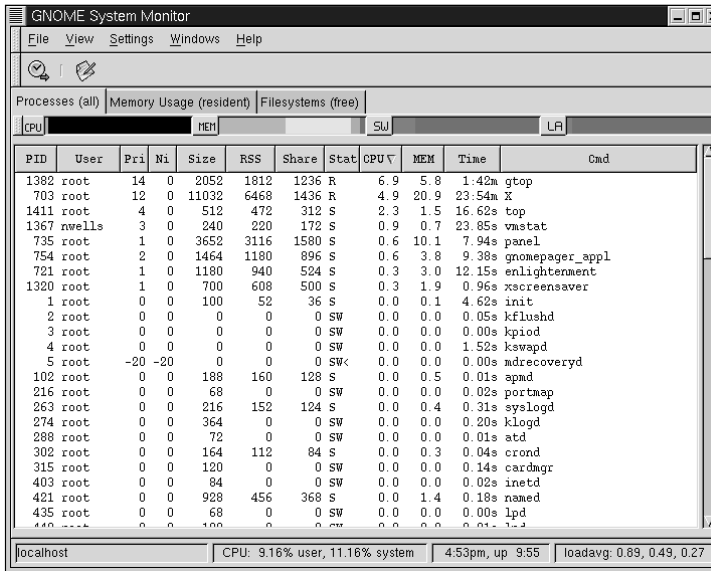
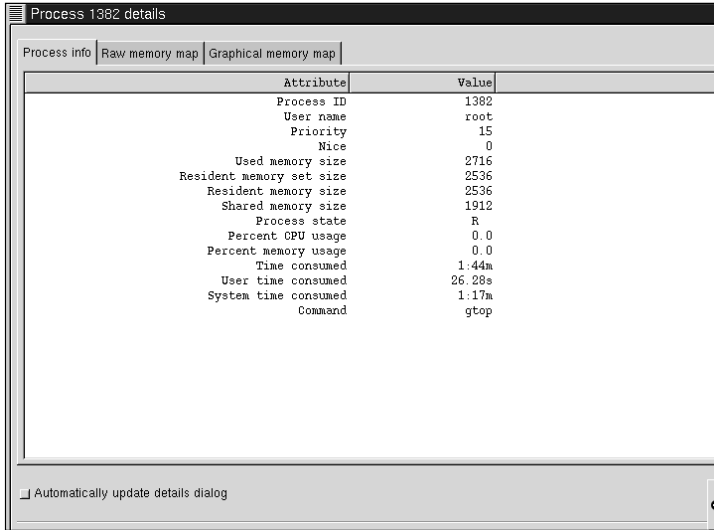


Figure 10-4 The Gnome System Monitor

The Gnome System Monitor display is very similar to those of the `kpm` utility and the `top` command. Graphs at the top of the window show CPU, memory, and swap space utilization. You can click on a process in the list to select it, then right-click to change the process priority or send a signal to the process.

The menu options in the Gnome System Monitor let you select which processes to display. To view or change settings for an individual process, right-click a process in the list, and then select a process option from the pop-up menu. Unlike the KDE Process Manager, the Gnome System Monitor provides details on how a single process is using CPU and memory resources. To view this information, click the process you want to display, right-click, and then click Details on the pop-up menu. This displays a details window like the one shown in Figure 10-5, which describes the CPU time consumed by the process with more precision than any of the programs discussed thus far in this chapter.

Figure 10-5 shows the `Process info` tab of the details window. Note that the details window also contains some additional tabs. Specifically, the `Raw memory map` tab shows memory usage numerically, while the `Graphical memory map` tab displays this information in graphical format. You will learn more about these two memory map tabs later in this chapter, when you learn about memory management.



Attribute	Value
Process ID	1382
User name	root
Priority	15
Nice	0
Used memory size	2716
Resident memory set size	2536
Resident memory size	2536
Shared memory size	1912
Process state	R
Percent CPU usage	0.0
Percent memory usage	0.0
Time consumed	1:44m
User time consumed	26.28s
System time consumed	1:17m
Command	gtop

☐ Automatically update details dialog

Figure 10-5 Details of a process in the Gnome System Monitor

Using Other Graphical CPU Status Tools

10

System administrators often use a simple graphical utility to track the CPU load of the Linux server. The KDE and Gnome utilities described in the previous section fill this need, but they also include other, sometimes unnecessary features. A basic utility called `xload` displays only a graph of the CPU load, with a higher line for heavy loads (during times of peak usage) and a lower line for light loads. Figure 10-6 shows the `xload` utility on a Linux desktop.

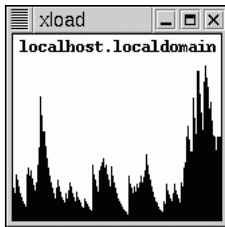


Figure 10-6 The `xload` utility

A system administrator can leave a window running `xload` open in order to see how the CPU is being used over time. This provides a graphical representation of potential problems when the load level in `xload` indicates that the CPU is continually under a very heavy load. By watching `xload`, the system administrator can decide when or if it is necessary to reduce the process load so that all users on the system experience better performance.

Utilities similar to `xload`, but designed specifically for KDE and Gnome, are also available. These utilities allow you to display a CPU load monitor graph in your tool panel or in a separate window. Many other CPU monitoring tools are available for Linux, though

the tools described here (such as `xload`) are the most commonly employed and are likely to be included with any Linux product you might be using.

Actively Monitoring the CPU Load

So far you have learned about many tools available for viewing process information and update the status of one or more processes. This section discusses ways to apply your knowledge of Linux process management to your system.

First of all, it's wise to keep a CPU load monitor such as `xload` visible on your desktop as you work. When the CPU load is consistently high, begin checking for processes that may need attention. When you begin working on a new Linux system, you may need to judge what level of load on the CPU equates to slow response times for users based on their reactions to different load levels. A very fast CPU can tolerate a high load level with acceptable response times, whereas a slower CPU cannot.

When you decide to investigate the cause of a high load, begin by using `top` (or a similar graphical utility) to see if a single process is using an inordinate amount of CPU time. A “run-away process” started by a user may be the single cause of the heavy load. In this case you can change the priority of the process, talk to the user who started the process, and kill the process if necessary. As you become more familiar with the normal load on your Linux system, you will be able to tell from the output of `xload` whether a single process has suddenly run wild (for example, because of incoming network traffic or a programming bug), or the overall load has grown to a higher level because of work done by many processes or users.

When many processes are causing the heavy CPU load, the `ps` and `top` command options (or the graphical utilities) can help you determine whether one set of programs (such as the Web servers), one type of program (such as shell scripts), or one user's programs are causing the heavy load. In each case, when you know the details, you can take corrective action by renicing a set of processes based on the command name or the username.

In some situations, everything may appear to be normal, with no processes taking undue CPU time and no troublesome applications running anywhere on the system. Such a “normal” system may nevertheless be very busy. When this situation persists for several days or weeks (depending on the IT strategy of your organization), you may have to begin planning for increased capacity. As a rule, you will always need more of everything in the future. Tracking CPU usage and taking action to correct errant processes simply lets you delay spending money on additional computing power until you really need it.

The CPU load can be reduced in many ways. This list shows a few of them:

- Raise the nice level of numerous user processes so that they are not all competing at the same level. Of course, users will complain about the slowness of the system unless the CPU is fast enough to run their applications adequately. You must determine the importance of various tasks and then judge the performance levels that are allowable for different users and tasks.
- Add a microprocessor (creating a dual-processor system). This can significantly reduce the load on your server, but it may require a complex upgrade to both your hardware and your Linux kernel.

- Move some tasks to a separate computer. For example, separate the Web server or e-mail server from the system used to store users' home directories and applications. The NFS protocol can be used to remotely access users' directories stored on another server.
- Add memory. This will often reduce the CPU load on a busy system because it obviates the need for the CPU to spend time moving data to and from the swap space.
- Use devices with higher performance, which can reduce CPU load by eliminating waiting time in the applications that access those devices. Examples of devices that can improve system performance include SCSI hard disks or CD-ROM drives (in place of IDE devices), faster network cards, and accelerated video cards.

Improving system performance by lowering the CPU load while running a given set of applications is a continual quest for Linux system administrators. The sections at the end of this chapter provide some additional guidance for improving performance by locating and removing system bottlenecks.

MANAGING MEMORY

The previous section described how to track and manage a system's CPU time. This section describes how to manage another key system resource: physical and virtual memory. Physical memory, also called **random access memory (RAM)**, is the electronic storage used by your computer for all operations. **Virtual memory** is the swap space that the kernel uses to store inactive processes.

More memory—both physical and virtual—always leads to better performance. This is true because the Linux kernel and Linux programs can only interact with information stored in memory. Information stored on a hard disk must be loaded into memory before it can be manipulated or presented to a user. Because memory is so valuable, you must manage it carefully to provide the best performance for users on your Linux system.

You can't manage memory in the same way that you manage CPU time. After all, a Linux process must have a certain amount of memory to operate. You cannot take away memory from a process in the same way that you can slow down the process by taking away CPU time (that is, by using a higher nice level). Instead, you must watch how memory is being used on the system. If you find that the processes running on your Linux system consistently need more memory than is available, you should increase the amount of memory on the system or change how processes are used.

Changing how processes are used means moving some processes to a different system or running them at a different time. This is not the same as changing how a single process uses memory, which you cannot do as a system administrator except for rare occasions when an application provides configuration options for such things.

Understanding Shared Libraries

When a developer writes a program for Linux, he or she uses a collection of functions called **programming libraries**. These libraries provide functionality that is common to many applications. By using the library, a developer is freed from the burden of re-creating common functionality. Each library contains numerous programming functions that a developer can refer to within a new program. A standard Linux system contains dozens of library files, each one holding a set of functionality for other programs to draw on.



Libraries in UNIX or Linux are similar to DLLs in a Windows program.

When you run a program that uses a library, the library must either be installed on your system or included with the program itself. Applications fall into one of two categories, depending on their relation to the library:

- **Statically linked applications** include the library functions in the main program. They require no additional library files on the Linux system. Each copy of an application loads a duplicate copy of all the library functions.
- **Dynamically linked applications** assume the library files are available on the Linux system. The library functionality is not included with the program itself. Dynamically linked applications use **shared libraries**. This means that several applications can use a single copy of the libraries that have been loaded into memory. (This also means that if the correct libraries are not loaded on the Linux system, a dynamically linked application cannot be used.)

Running multiple applications that are dynamically linked to the same libraries requires less memory than running multiple statically linked applications. To understand why, you first need to understand that the total memory allocated for a newly loaded application includes the memory required to store any shared libraries used by the application. This means that if you load one application and its attendant libraries, and then load a second application that is dynamically linked to those same libraries, the libraries will not be loaded a second time. Instead, the second application “shares” the libraries with the first application. If 10 applications were using one shared library, the library would still only load into memory once, rather than 10 times. Thus, dynamically linked applications save a great deal of memory in situations where applications use the same libraries.

Most Linux applications are dynamically linked, so they use a set of shared libraries that are installed on a Linux system by default. For example, if you start numerous KDE applications, most of the functionality of the application is contained in a single set of libraries shared by all applications.

Understanding Paged Memory

New Linux administrators sometimes have the mistaken impression that information is moved to and from swap space one application at a time. You may assume, for example, that as physical memory becomes full, an inactive application will be copied to the swap partition in order to free memory for a newly activated program. In fact, however, information is transferred to and from swap space in smaller units, known as pages. A **page** of memory is a block of 4 KB of RAM.

Whenever an application requires additional physical memory, the Linux kernel copies pages of memory to the swap partition. It does not copy an entire application; rather, it copies only enough pages to free the amount of memory needed by the newly activated application. The copied pages might be taken from the middle of the inactive application's memory. The kernel keeps track of which pages of memory are moved to the swap partition. The free space created by moving these pages to the swap partition is then made available for other programs that need physical memory immediately.

When a program that was inactive becomes active again, the kernel copies the swapped pages back from the swap partition to the same place within the memory space used by the application. Because the memory is restored by the kernel, the application cannot tell that its memory was used by another application for a time.

Swapping individual pages of memory (rather than complete applications) dramatically improves the performance of Linux on heavily loaded servers. If Linux swapped complete applications (rather than individual memory pages), system resources might be wasted copying a very large application to the swap partition when in fact only a small percentage of its memory was needed by another program.

10

Tracking Overall Memory Usage

Within the realm of memory management, your most important job is making certain that your Linux system has sufficient memory for the applications users want to run. To view the status of the memory, use the **free** command. A typical command would be as follows:

```
free
```

The output of the **free** command is shown here:

	total	used	free	shared	buffers	cached
Mem:	30820	30084	736	29960	656	10876
-/+						
buffers/cache:		18552	12268			
Swap:	130748	14144	116604			

All the information displayed by **free** is in kilobytes. You can use command-line switches to change the display to bytes or megabytes. The following list describes each of the columns of information in the output of the **free** command:

- The **Mem** line refers to physical memory (RAM).
- The **Swap** line refers to swap space (located on your swap partition).

- The `total` column indicates the total amount of memory available to Linux. The sample output shows a system with about 30 MB of RAM and 128 MB of swap space.
- The `used` column indicates how much of the total memory is currently in use. Both RAM and swap space include an amount of used space.
- The `free` column indicates how much space is free. Both RAM and swap space include a free space amount.
- The `shared` column indicates how much of the used space is dedicated to shared libraries. Often this will be the majority of the used space.
- The `buffers` column indicates the amount of **buffers** memory, which is dedicated to data storage for the applications that are running. This number will be large if you are running applications that work with data files, such as word processors or spreadsheets.
- The `cached` column indicates memory that is being used by the Linux system for cached data (information from the hard disk that is stored temporarily in RAM, under the assumption that it will soon be needed by an application). If few applications are running, Linux will use most of the available RAM as a cache to improve performance. As more applications are launched, less memory is used for caching.

The following list describes some indicators to watch for in the output of the `free` command:

- If the value in the `shared` column is small on a busy system, you may be able to decrease memory consumption by using more applications that are dynamically linked. Check the documentation for large or critical applications to see if a dynamically linked version is available.
- If the last line of the `free` column (the `Swap` line) is small, you are in danger of running out of both physical and virtual memory. This situation can cause the system to crash under a heavy load. At the very least, it will decrease performance significantly as applications wait for memory to become available.
- If many applications are being run at the same time on a system with a small amount of physical memory, they will be continually moved to and from the swap space. This situation, called thrashing, wears out the hard disk mechanism and significantly decreases application performance. Thrashing indicates that you need more physical memory (RAM) to run the applications you intend to use.
- The first line of the `free` column (on the `Mem` line) is normally very small. This is because the Linux system tries to use all available RAM for caching hard disk information. If applications are launched, less information will be cached. Thus, if the `Mem` line of the `free` column indicates a value near zero, you may still start several applications without using a significant amount of swap space. Check the value of the `cache` column to see how much memory might be available for additional applications.

In addition to using the `free` command to learn the status of system memory, you can use fields in the `ps` command. Several fields in the output of the `ps` command provide information about a process's memory usage:

- The `%MEM` field shows the percentage of available system memory that the process is using.
- The `STAT` field (for Status) shows whether the application is sleeping (indicated by an `s`). Sleeping applications can be swapped to hard disk if the memory they occupy is needed by another process. The `STAT` field also shows a `w` if part of the process is already swapped to hard disk. The command line naming the process (the `COMMAND` field) will be enclosed in square brackets if part of the process has been swapped to hard disk.
- The `RSS` (Resident Set Size) field shows the amount of RAM currently used by the process. This value is given in kilobytes.

The Gnome System Monitor, mentioned earlier in this chapter, provides several helpful graphs that you can use to track system memory usage. The Gnome System Monitor is launched from the `utilities` menu in Gnome (or by entering the command `gtop`).

After opening the System Monitor window, you display the middle tab by clicking `windows` on the menu bar and then clicking `Memory Usage`. Figure 10-7 shows the resulting memory usage graph.

10

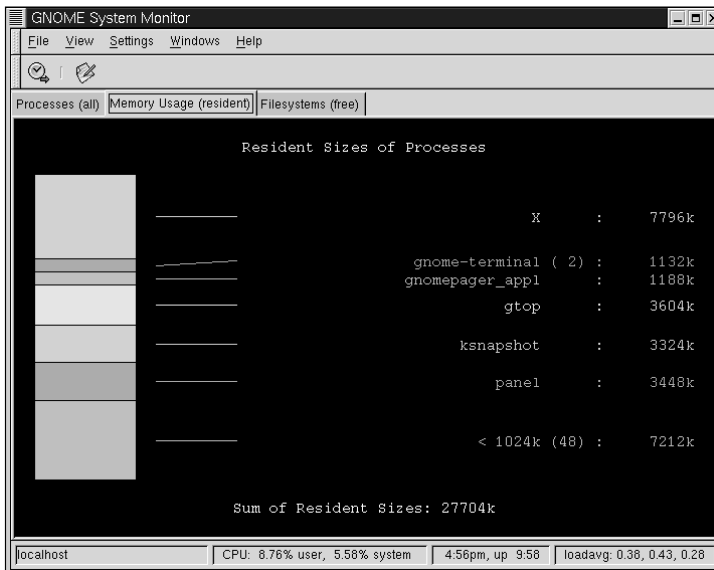


Figure 10-7 Memory usage displayed by the Gnome System Monitor

The process that uses the most memory is shown at the top of the graph. Additional applications are listed until the bottom of the window is filled. A summary of processes below the threshold value (1 MB by default) indicates the total memory used by those processes. A

summary of the amount of memory used by all processes is shown below the graph with the label “Sum of Resident Sizes.”

By default, this window shows the amount of RAM used by each process. With the **Memory Usage** tab displayed, you can use the **view** menu to display:

- **Shared Sizes of Processes** (usage of shared library memory space)
- **Virtual Sizes of Processes** (including the actual size of the program and the sum of all shared libraries the program uses)
- **Swapped Sizes of Processes** (the amount of space used by each process on the swap partition)
- **Total Sizes of Processes** (comprising the resident memory, or RAM, used and the swap space used)



In the **Preferences** dialog box (available via the **Settings** menu) you can also select the threshold level below which process names are not included on the graphs for each type of memory display.

Tracking Per-Application Memory Usage

You can use the graphs in the Gnome System Monitor to learn how a specific process is using memory. This information is included in the memory usage screen (see Figure 10-7) if the process is one of the larger memory consumers on the system. By viewing the details of one process, however, you can see exactly how that process uses memory, even if it is a small process consuming very little memory. Furthermore, by reviewing information about specific processes that are run frequently on your system, you may learn how to use your system resources more efficiently.

Within the Gnome System Monitor, view the process list by choosing **windows** on the menu bar and then choosing **Processes**. Click on a process, and then right-click to display a pop-up menu of options. From this menu, choose **Details**. The **Details** dialog box for the process you selected appears. The **Process info** tab of this dialog box is shown in Figure 10-5. This tab contains information about the memory used by the process, including the resident memory size (amount of RAM the process is using) and the shared memory size (amount of memory used by shared libraries that this process relies upon to function).

Additional information about how a process uses memory is available on the **Raw memory map** and **Graphical memory map** tabs. The **Raw memory map** tab, as illustrated in Figure 10-8, shows all of the library files used by the process. A great deal of detailed information is provided on this screen. The information here can be used to diagnose problems with an application as well as to see how the application is using system memory.

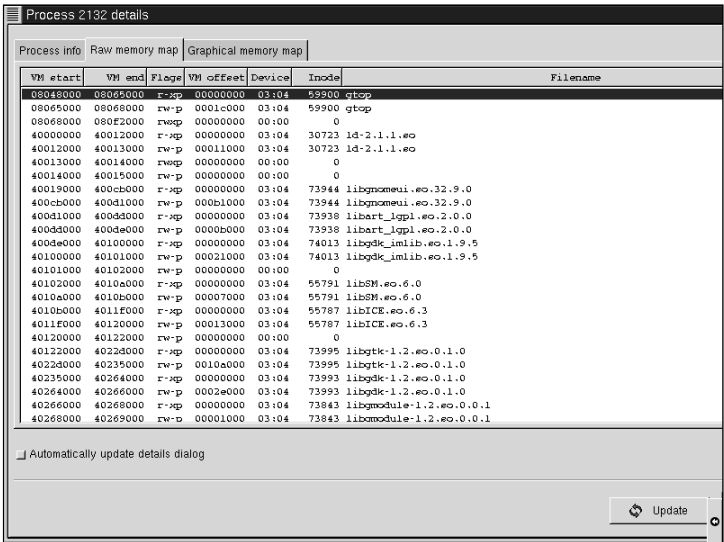


Figure 10-8 The Raw memory map tab in the Gnome System Monitor

The Graphical memory map tab, shown in Figure 10-9, provides information that is more useful to nonprogrammers. By reviewing the information on this tab, you can see the relative size of each section of the selected process. The program itself, as well as all of the libraries it uses, is divided into program code and data sections. The graph on the left of the window shows the relative sizes of each library used and any data files currently being used.

10

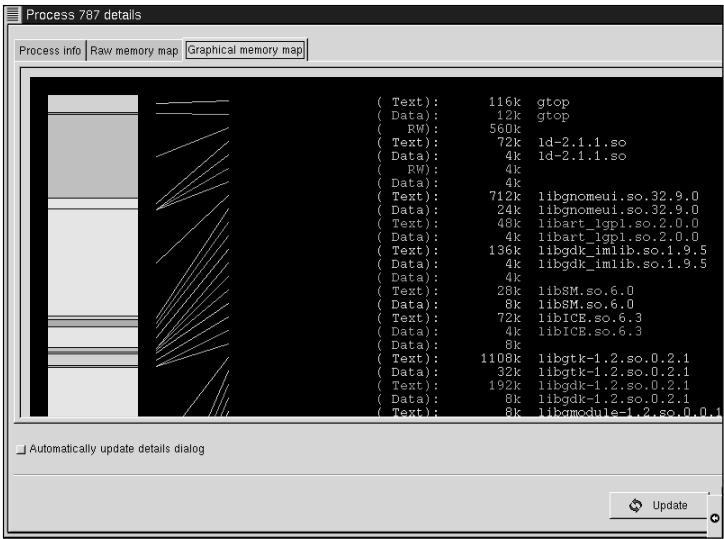


Figure 10-9 The Graphical memory map tab in the Gnome System Monitor

Viewing Virtual Memory Information

The general state of the swap space (the virtual memory) on your system is shown by the `free` command and by both graphs and numeric displays in several of the utilities described in this section (such as `kpm` and Gnome System Monitor).

You can use the `vmstat` command to examine detailed information about how the swap space on your system is being used. The output of the `vmstat` command is cryptic until you become familiar with the abbreviated labels for the fields that it displays. When `vmstat` is run as a regular command, its output is based on information averaged over time since the system was booted. You can also run `vmstat` as you would `top`, with updated information provided every few seconds. In this case the information is computed since the last update rather than since system boot time. To run `vmstat` as a regular command, simply type `vmstat`. Sample output is shown here:

```
procs      memory      swap      io      system      cpu
r b w  swpd  free buff cache  si so  bi bo  in cs  us sy id
0 0 0   11112 4108 700 11980   1 1   6 1  119 87   3 3 94
```

The fields displayed in the output of `vmstat` are explained in the following lists. Under the `procs` main heading:

- `r`: Number of processes waiting for run time
- `b`: Number of processes in uninterruptible sleep
- `w`: Number of processes swapped out but otherwise runnable

Under the `memory` main heading:

- `swpd`: Amount of virtual memory used (KB)
- `free`: Amount of idle memory (KB)
- `buff`: amount of memory used as buffers (KB)
- `cache`: amount of memory used to cache hard disk data (KB)

Under the `swap` main heading:

- `si`: Amount of memory swapped in from disk (Kbps)
- `so`: Amount of memory swapped to disk (Kbps)

Under the `io` main heading:

- `bi`: Blocks sent to a block device (the hard disk), measured in blocks per second
- `bo`: Blocks received from a block device (in blocks per second)

Under the `system` main heading:

- `in`: Number of interrupts per second, including the clock
- `cs`: Number of context switches (changes between active processes) per second

Under the `cpu` main heading each item indicates a percentage of total CPU time for three areas:

- `us`: User time
- `sy`: System time
- `id`: Idle time

To run the `vmstat` command in interactive mode as you would `top`, so that its output is periodically updated, you need to type a number after the command indicating the delay between updates. For example, the following command displays an updated line of information every two seconds:

```
$ vmstat 2
```

The information provided by `vmstat` is useful for locating bottlenecks on your system related to hard disk performance, lack of sufficient memory, or problems with specific applications. The next section discusses these issues in some depth.

LOCATING SYSTEM BOTTLENECKS

A **bottleneck** is the part of a computer system that significantly slows down completion of the task at hand. To understand the bottleneck metaphor, think of holding a bottle of soda pop upside down and trying to pour the contents out as rapidly as possible. Because the neck of the bottle (the bottleneck) is narrow, the liquid is restricted; it can't flow out as rapidly as it would if you cut off the top of the bottle, removing the bottleneck. Computer systems always have a bottleneck—a part that is slower than the other components of the system. Your goal as a system administrator is to make certain that the bottlenecks are large enough or hidden enough that they do not hamper the organization's computer system needs. When you discover that a computer system is not performing adequately, you must identify the bottleneck and improve the situation. Here are several examples:

- Many users on the network are accessing the Internet at the same time over a single dial-up modem connection. You all complain about slow service. The modem-based network connection is a bottleneck. To fix it, you could change to a higher-speed Internet connection using another technology.
- A large database index is created automatically each morning at about the time users are arriving for work. You notice that the system response is slow until the database task is completed. The CPU capacity is a bottleneck. You could add a separate CPU for the database to run on, add memory to improve database performance, or schedule the database task to begin earlier, so it does not compete with users' tasks each morning.
- A special-purpose server gathers network statistics and stores them on the server's hard disk. Because the server's hard disk is old and slow, and the network is very fast, the server generates statistics from the network faster than they can be written to disk, resulting in lost data. The hard disk speed is a bottleneck. To fix it, upgrade the hard disk to a newer, faster model with a faster interface, such as SCSI.

These three examples illustrate that bottlenecks in your computer systems can be inside your computer or outside—usually in the network infrastructure. Depending on the tasks your server is dedicated to, just about any part of your Linux server could become a bottleneck. In other words, possible bottlenecks include:

- Physical memory (RAM)
- Swap space (virtual memory)
- CPU time
- Hard disk space
- Hard disk access speed
- Video card speed
- Serial port speed
- Network card speed
- Internal bus speed



Internal buses such as ISA, EISA, MCA, and PCI determine how rapidly data can be moved between components of a single computer, such as from the system's RAM to its hard disk.

Identifying and Removing Bottlenecks

You can use the tools described in this chapter to identify and eliminate bottlenecks on your Linux system. Before you start, however, you must realize that improving the performance of a Linux system is to some degree more art than science. It is difficult to understand exactly what is happening in the system, but you must develop an understanding before you can make adjustments that will noticeably improve performance for yourself and for end users.

The only way to master this art is by working with Linux systems and related technologies for a long time. In the beginning, however, you will need to rely on science—that is, using various tools, testing many programs, and reviewing pages of numbers that indicate performance measurements and system status. These sometimes laborious steps will eventually lead you to a more intuitive understanding of the way events unfold within the system. At that point, you will probably use the commands described in this chapter to verify your own judgments, rather than to locate the source of a problem.

Because networking technologies are a major cause of bottlenecks on a Linux system, you will not be able to fully optimize your system until you have also learned something about networking with Linux. However, in the meantime, you can extract a great deal of helpful information about your system from the `free` command and from the various graphical monitoring tools. Even without extensive knowledge of networking, you can use these tools to locate problems related to poor system performance.

Using Benchmarks

One additional tool that can help you identify bottlenecks is a **benchmark program**, which provides a numeric measurement of a part of a system's performance. You can use the benchmark value to compare multiple systems that are working on the same task (for example, running the same program or responding to Web requests). A benchmark is also useful for showing you the speed at which events unfold within your system. If that speed is lower than expected, you can then look for the bottleneck that is slowing performance. The following list describes some benchmarks that you can use to test the performance of your Linux system:

- The values given in `/proc/cpuinfo` include a field called `bogomips`, which provides one measure of the speed of your processor. Note that this value is not a highly accurate measure of the CPU speed on your system (hence the name, which is short for “bogus measure of instructions per second speed”). This value on a single machine is not very useful, but comparing it among multiple machines can help you determine whether the CPU in your Linux system is slower than you thought it was.
- The **nbench** program measures CPU speed by running tests on tasks such as mathematical computations, sorting, and data compression. The **nbench** program is a Linux port of the benchmarks developed by now-defunct *BYTE* magazine. It is actively maintained at http://members.xoom.com/Gavrilov_Con/nbench/index.html.
- A hard disk benchmarking tool called **bonnie** provides throughput statistics in megabytes per second for various types of disk transfers, such as random and sequential reads. This program is an excellent way to identify bottlenecks in your system's performance. You can learn more about the **bonnie** package and link to the free download at <http://www.textuality.com/bonnie/>.

CHAPTER SUMMARY

- The Linux kernel makes a great deal of system information available via the `/proc` file system. This information is used by many Linux utilities that help system administrators manage processes and system memory in order to improve performance.
- To manage processes effectively, you must rely on information from programs like `ps` and `top` to identify how much CPU time a process is using. The utilities `nice` and `renice` can be used to change the CPU time that a process is allowed to consume. Graphical utilities can make process management more convenient.
- When managing memory, it's important to keep in mind that all applications must compete for a finite amount of system memory. But by watching the use of physical and swap memory (via utilities like `free`), a system administrator can determine whether memory in a Linux system is a bottleneck that degrades system performance.

- Other standard graphical and text-based utilities, as well as benchmark programs, can be used to identify bottlenecks in a busy Linux system. Identifying and removing bottlenecks to increase system performance is a challenging task.

KEY TERMS

benchmark program — A program that provides a numeric measurement of performance for part of a system.

bonnie — A hard disk benchmarking program.

bottleneck — Part of a computer system that slows down completion of the task at hand.

buffers — Areas of memory dedicated to holding the working data associated with a running program.

cached data — Information from the hard disk that is stored temporarily in RAM, under the assumption that the data will soon be needed by an application. Caching data improves system performance, because data can be accessed faster from RAM than from the hard disk.

dynamically linked applications — Linux programs that do not include the library functions that they require in order to operate. The libraries must be installed (as shared libraries) on the Linux system on which the applications are executed.

file handle — An internal storage mechanism that allows a single file to be opened and used in Linux.

free — Linux command used to display the amount of free and used memory (physical and virtual), with basic information about how that memory is being used.

Gnome System Monitor — A graphical utility for the Gnome Desktop that is used to monitor and control processes running on Linux. Also called `gtop`. Similar to the `kpm` program for KDE.

kpm — The KDE process manager, a graphical utility for the KDE graphical environment that is used to control Linux processes. Similar to the Gnome System Monitor.

ktop — A KDE-based graphical utility used to manage Linux processes. Similar to the `kpm` program.

nbench — A benchmark program developed by *BYTE* magazine and ported to Linux.

nice — Linux command used to set the nice level of a Linux process as it is being launched.

nice level — The priority level assigned to a Linux task.

page — A block of 4 KB of memory. A page is the unit of memory in which the Linux kernel moves data to and from the Linux swap partition.

priority — A value assigned to a process running on Linux that determines how much CPU time is granted to the process to complete its tasks.

/proc file system — A method of viewing what the operating system kernel is doing at all times, using a section of the Linux directory structure and a file system interface (using the same commands used for regular files).

programming libraries — Collections of programming functionality that are common to many applications. *See also* shared libraries.

- random access memory (RAM)** — Electronic storage used by a computer as a working space for all operations while the computer is turned on.
- renice** — Linux command used to change the nice level of a Linux process that is already running.
- shared libraries** — Programming libraries that are used by several dynamically linked applications running at the same time. Shared libraries are used to reduce memory consumption by providing a single copy of redundant functionality.
- statically linked applications** — Linux programs that include library functions in the program itself so that they are not dependent on the libraries loaded on the Linux system.
- top** — Linux command used to view the most CPU-intensive processes running on Linux at a given moment, along with related process information for those processes.
- virtual memory** — The swap space on a Linux system. Stored in a magnetic or other nonvolatile location, such as a dedicated hard disk partition.
- vmstat** — Linux command used to display detailed information about virtual memory usage.
- xload** — A graphical program that displays the CPU load over time on any Linux system's graphical interface.

REVIEW QUESTIONS

10

1. Although the `/proc` file system is not stored on a hard disk, it is called a file system because:
 - a. It is accessed via a directory structure using standard commands that also apply to regular files.
 - b. It contains information about partitions, mount points, and hard disk devices.
 - c. The information from `/proc` can be stored on a hard disk using standard redirection operators.
 - d. Special utilities must be used to access the information contained in `/proc`.
2. For some system parameters, information can be written to the `/proc` file system. True or False?
3. The command `cat /proc/partitions` will display which of the following:
 - a. A list of all swap partitions that the kernel is actively using
 - b. A list of all partitions known to the Linux kernel
 - c. The raw contents of the partition table for the `root` file system
 - d. The equivalent of the Linux `mount` command output
4. Name five commands that rely on the information provided by the `/proc` file system to create their output.
5. Only the `root` user can view processes started by all users. True or False?

6. The command `ps -A xo comm` will display the following information:
 - a. The command is invalid; it will display an error message.
 - b. All processes started by the user who executes the `ps` command, with a standard set of fields pertaining to that user.
 - c. All processes will be listed; a revised nice value will be requested for each one that matches the string `comm`.
 - d. All processes running on the system, with the nice level and command-line fields displayed for each one.
7. The _____ field in the `ps` command output defines how much cumulative CPU time a process has used since it was launched.
 - a. TIME
 - b. %CPU
 - c. RSS
 - d. START
8. The `CMD` field of the `ps` command output displays:
 - a. The command used to start the `ps` command (including all `ps` command options applied to the current output)
 - b. The command used to start the process shown on each line
 - c. The last signal sent to control the process on each line
 - d. The equivalent output from the `vmstat` command
9. The `root` user can run the `renice` command to increase or decrease the priority level of any current process. True or False?
10. Which of the following commands is invalid if run by a regular user?
 - a. `renice -10 1035`
 - b. `renice 10 1035`
 - c. `vmstat 5`
 - d. `renice 5 1035`
11. Which of these programs allows you to change the nice level of a running process (choose all that apply)?
 - a. Gnome System Monitor
 - b. `free`
 - c. `kpm`
 - d. `top`
12. To update the process data displayed by the `top` command, you would press which key?
 - a. r
 - b. Spacebar
 - c. u
 - d. n

13. Name three graphical system management tools. Explain on which platforms they can be used and how to start each one.
14. Memory is not managed the same way that CPU time is managed because:
 - a. Memory is a physical resource; CPU time is not.
 - b. CPU time can be allocated restrictively among processes, slowing them down; memory cannot—a process must have its required memory allocation.
 - c. The set of utilities available for managing memory is smaller and less effective than the toolset available for process management in Linux.
 - d. Virtual memory is stored on a hard disk, which is a CPU-intensive activity that limits configuration options.
15. Shared libraries are commonly used in Linux applications. True or False?
16. Dynamically linked applications are preferred for their better memory usage unless:
 - a. Statically linked applications are also available.
 - b. Multiple users need to run the same application at the same time.
 - c. The necessary libraries to run the application are not installed on the Linux system.
 - d. The `free` command indicates that only virtual memory is available.
17. Contrast the output and use of the `free` and `vmstat` commands.
18. Explain why the amount of free RAM on a Linux system, as shown by the `free` command, is generally very small.
19. Thrashing occurs when:
 - a. An excessive amount of information is moved to and from the swap partition in a short time.
 - b. The `top` and `ps` commands both try to access process information at the same time.
 - c. Physical and virtual memory are deadlocked over where program data should be stored.
 - d. Multiple bottlenecks limit the speed of a Linux system.
20. A bottleneck occurs in a Linux system when:
 - a. The system administrator is not able to respond to all end-user requests in a timely fashion.
 - b. The `free` command indicates a shortage of physical memory.
 - c. One part of the system restricts adequate system performance because it cannot keep up with the demands of other system components and applications.
 - d. Multiple processes try to access the hard disk at the same time.

21. The `vmstat` command is useful for identifying bottlenecks because:
 - a. It correlates precisely to the swapped process information given in the `ps` command output.
 - b. In conjunction with the `bonnie` benchmarking tool, it provides detailed memory speed information.
 - c. It can be used only by the `root` user, thus preventing interference by regular users consuming system resources.
 - d. It provides detailed information about the number of processes waiting for access to physical memory.
22. Name seven parts of a Linux system's hardware that could become bottlenecks. For at least three of them, describe the situation in which that component would be a bottleneck.
23. Which statement best describes the process of improving performance on Linux systems?
 - a. The art and science of it are best left to end users.
 - b. It requires careful study of system details and user needs.
 - c. It is only possible to make educated guesses about ways to improve system performance.
 - d. With a Linux system, performance is usually fixed and cannot be affected by adjustments made by the system administrator.
24. The `bonnie` program is useful for determining which of the following:
 - a. Hard disk performance data
 - b. CPU speed benchmarks
 - c. The balance of virtual and physical memory
 - d. The status of information in the `/proc` file system
25. The command `vmstat 4` will do which of the following:
 - a. List information on the process with PID of 4 if it is located in virtual memory.
 - b. Set the default nice level for virtual memory processes to 4.
 - c. Display continuous updates of the virtual memory status on a new line every four seconds.
 - d. Start four instances of the virtual memory management module.

HANDS-ON PROJECTS



Project 10-1

In this activity you explore the `/proc` file system and see how your actions affect the information returned by queries to a filename in `/proc`. To complete this activity you need an installed Linux system.

1. Start a command-line window in Linux.
2. Change to the `/proc` directory.
3. Display the contents of the `/proc/mounts` file with the command `cat /proc/mounts`.
4. Insert a formatted floppy disk in the disk drive and mount it using this command: `mount -t msdos /dev/fd0 /mnt/floppy`. (The mount directory may be different depending on the configuration of your Linux system. You may also need to be logged in as `root` to mount the floppy disk. If the disk is a Linux-formatted disk, substitute the file system type `ext2` for `msdos`.)
5. Execute the command `cat /proc/mounts` a second time. Is the floppy disk device shown?
6. Execute the command `mount` and compare its output with the output from Step 5. What additional information does the `/proc` file system include? Do you recognize that information from a system configuration file located in the `/etc` directory? What are the advantages of using `mount` instead of reading the `/proc` file system?

10



Project 10-2

In this activity you use the Gnome System Monitor to explore different ways of viewing process information on your Linux system. The tasks shown in this activity could also be done using the `ps` command on any Linux command line. The graphical interface provides a good way to interact with a large amount of system information without requiring you to memorize numerous command options. To complete this task you should have a copy of Linux with the Gnome Desktop installed (this would probably be a copy of Red Hat Linux).

1. Log in to Linux and start the graphical Gnome Desktop environment. Next, you need to start a program that you can monitor.
2. The Gnome Panel is the toolbar at the bottom of the Gnome screen. Click the footprint icon on the left end of the Gnome Panel to open the main menu, point to **Utilities**, then click **Simple Calculator**. Now you can use the Gnome System Monitor to monitor this program.
3. Click the Gnome Panel to open the Gnome main menu again. On the main menu point to **Utilities**, and then click **System monitor**. (You can also enter `gtop` in a terminal emulator window.) The main window of the System Monitor utility appears.

4. Click **view** on the menu bar, and then click **Only TTY Processes**. This limits the list of processes to those associated with one of the Linux terminals. Can you identify a few processes that are removed from the list when you choose this option?
5. Click **view**, then click **Only TTY Processes** to deselect that option. Click **view** again, then click **Hide System Processes**. How would you describe the process list now?
6. Select a process in the list by clicking it. Then right-click on that process and click **Renice** in the pop-up menu. Raise the nice level of that process to 20 with the slider bar and click **OK**. You see the new nice value in the **NI** field. Can you also see the CPU percentage field changing? (You will not notice a change in the CPU percentage field if the value of the CPU percentage field started as 0.0 before you changed the nice level of the process.) Change the process back to a nice level of 0 before proceeding.
7. The calculator process that you started in Step 2 is a program called **gcalc**. Locate that process in the list, right-click on it, and then click **Details** in the pop-up menu. The details window for the selected process opens. If necessary, click the **Process info** tab to display it.
8. Select the check box **Automatically update details dialog**.
9. Display the window for the selected process again. Begin using the process. Can you see the information on the **Process info** tab change as you use the process?
10. Click the **Graphical memory map** tab of the details window.
11. Switch back to the window of the application whose details you are viewing. What actions in the application cause the Graphical memory map to change?



Project 10-3

In this activity you track the status of the virtual memory usage (swap space) as you work with several applications. Normally you will not track virtual memory this carefully as you personally work with applications. Instead, you will have the easier job of watching the system-monitoring tools while other users are working with the programs. To complete this activity you should have an installed Linux system. The Netscape Communicator program is used in the steps that follow, but you can substitute any large program on your system.

1. Log in to your Linux system and start the graphical system.
2. Open a command-line window.
3. Use the following **vmstat** command in interactive mode to display an updated status line once every two seconds: **vmstat 2**.
4. Click the Netscape icon on your desktop (or open a second command-line window and enter the command **netscape**).
5. Watch the values in the **vmstat** output change as the Netscape program starts. Which fields do you see changing? Do you see any changes that indicate a potential bottleneck on your system if many copies of Netscape were started at the same time?
6. Open a second terminal emulator window and enter the following command to see the physical and virtual memory information, updated once every two seconds: **free -s 2**.

7. Start another large program, such as a second copy of Netscape, a spreadsheet, or a large system administration tool such as LinuxConf, the KDE Control Center, or the Gnome System Monitor.
8. Watch the values in the output of the `free` command change. Do the values in `vmstat` change as well? What additional information can you see in the `vmstat` output? If you opened a second copy of Netscape, how do the shared libraries used by the two copies of the same program affect the memory usage when you started a second copy?
9. Click on one of the command-line windows (where you are running `vmstat` or `free`) to activate the window. Then press **Ctrl+C** to end the output of the command displayed in that window. Enter the command `top` and review the output to see which processes are currently swapped out (stored on the swap partition). The letter `w` in the `STAT` column indicates that a process is swapped out.
10. As you read the output of these system administration tools, what can you conclude about how these tools affect system performance? What comments would you make about the value of using graphical system-monitoring tools such as those mentioned in this chapter?

CASE PROJECTS

1. You have been called in as a consultant to Home Care, an insurance company that runs a busy Web server for customers and numerous internal computers for employee use. You have been asked to solve a problem with the company's Web server. Although the company has a very fast Internet connection (T3 speed—45 Mbps) to service the thousands of customers who access the site, customers regularly complain about the slow speed of the Web server, particularly when requesting account information. You recall having installed very high performance SCSI hard disks on the Web servers. Considering the occasions when customers report the slowest service (during account inquiries), what might you conclude about the bottleneck that slows down Web server responses? How could you test your theory using some of the utilities you learned about in this chapter? How could you solve the problem using some of the methods described in this chapter? If the slow performance were a problem on all Web server requests, where might you look outside of the Home Care network infrastructure to identify potential problems?
2. Employees at Home Care access company data through a second Linux server that is unrelated to the public Web server. But they also complain about generally slow performance as they work. What tools could you use on the Linux server to determine where the drag on system performance originates? Keeping in mind that more than 100 employees access the Linux server during the day over a standard 10 MB Ethernet network, what are some possible bottlenecks in the network infrastructure or the server? Could the problem lie with each employee's Windows client? If so, how might you demonstrate this to management? Assuming the bottleneck was on the Linux server, how would your initial capacity planning (see Chapter 7) affect the ease and cost of upgrading the Linux server to handle employee needs with an acceptable level of performance?

